Datasheet management

someonesdad1@gmail.com 2 Jun 2014

Package needs: ds.py, abspath.py, _goto.py

I keep a directory on my computer called `manuals` that has the following subdirectories: `app_notes`, `catalogs`, `datasheets`, and `manuals`. Here are a few of the files in these directories:

```
./app_notes/AC_power_source_analyzer.pdf
./app_notes/DMM_errors_1.pdf
./app_notes/HP_Journal_Jan1964_3400A.pdf
./catalogs/06_PomonaCatalog.pdf
./catalogs/2007MeyerGageCatalog-2.pdf
./catalogs/Agilent_multimeters.pdf
./datasheets/1N4148_diode.pdf
./datasheets/2N2222_npn_transistor.pdf
./datasheets/batteries/EN_AA_alk_battery.pdf
./manuals/Bosch_laser_rangefinder.pdf
./manuals/Dixon_lawn_mower_ZTR3362/1997_mower_manual.pdf
./manuals/Wavetek_144.pdf
./manuals/calculators/hp15c.pdf
```

Over the years I've collected around 1000 data sheets, manuals, and catalogs. When I'm working on something interesting, it's a pain to have to stop and go look for a manual or datasheet. This led me a number of years ago to write the `ds.py` script, which is a python script that will search the manual and datasheet directories for files matching a particular regular expression. For example, if I want to see the manual for my HP 3400 RMS voltmeter, I type the command[1] `ds 3400` and I'm prompted as follows:

```
--> ds 3400
Choose which file to open:
  1  app_notes/HP_Journal_Jan1964_3400A.pdf
  2  manuals/hp/3400A_catalog.pdf
  3  manuals/hp/3400A_RMS_voltmeter.pdf
?
```

I'm presented with a list of matches (the matching text is highlighted) and I type the number in for the particular document I want to see. In this case, I'd type in `3`. Then the `manuals/hp/3400A_RMS_voltmeter.pdf` file would be opened. This is done in conjunction with an `app.exe` program that launches a file with its registered application. See the appendix ***Appendix 1: app.exe*** below for how to build it if you don't want to use the Windows executable I include in the package.

When there's only one match for a file, it is automatically opened. It's fast and requires little thought, so you'll probably find you use it quite a bit. Of course, it can be used to find and open any set of files that have a registered application, so it's not limited to e.g. PDF files. I use a similar technique to open various project files and documents I'm working on.

The last time I was on a Linux box I was able to do similar things, although there's a way of launching things with the registered application directly without needing to compile a program (I've forgotten the details). Thus, you may have to do a small amount of hacking on the `ds.py` script to get it working on your machine, but it will be worth it because it will save you time browsing for files. This is one of those situations where it's faster using a console script than e.g. opening up an Explorer window and looking for the file you want.

I also use a copy of this script named `eb.py` to find various ebooks I keep on my system. I have thousands of ebooks and I just have to remember to give their file names descriptive titles. These

1 I use an alias or shell script to launch the python file.

two utilities (really, they're the same basic script) save me a lot of time from having to look for things. Usually, when I'm looking for a manual on e.g. an instrument, I just type `ds model_number` and I'm reading the manual file in about half a second after I type the command. There's no GUI tool that would be anywhere near as fast because you'd have to move your hand to the mouse and find an icon somewhere.

What you need to use it

The coloring of the screen is done by a python module called `WConio`. You can download the needed stuff from http://newcenturycomputers.net/projects/wconio.html. The installation will put two needed files into your python `Lib/site-packages` subdirectory (along with an `egg` file): `WConio.py` and `_WConio.pyd`. I rename `WConio.py` to `wconio.py` so I can use import `wconio` in my scripts.

If you don't want the color highlighting, it's easy to disable it in the `ds.py` script. Delete the import of `wconio`, comment out the lines containing color information, and comment out the functions `SetColors` and `fg` and the calls made to `fg` in `PrintMatch`.

How to set up the script

The primary task you need to do is hard-code the directories you want searched for the files. To do this, you'll need to edit the `d["dir"]` tuple defined in `main()`.

The script will find all the files under the indicated directories. If you want to restrict the files to a particular type, you can edit the globbing patterns in the `GetFiles()` function. For example, to only list text files and PDF files, you could use the following code:

```
f = [Normalize(i) for i in glob.glob(dir + "/*.txt")]
f += [Normalize(i) for i in glob.glob(dir + "/*.pdf")]
```

Note that the script only searches the indicated directories, not their subdirectories. It wouldn't be hard to add a recursive walk of each directory using `os.walk()`, but I wanted responsive behavior, so I limited the search to directories I knew contained the relevant files.

Once you've compiled the `app.exe` program (or use the executable I supply), you'll have to hard-code the path to it in the `ds.py` script. Near the bottom of the file are two lines containing `subprocess.Popen`; you'll want to change my `d:/don/bin/bat/app.exe` to your own path.

Appendix 1: app.exe

If you're on a Windows system, you'll need to use the `app.exe` program to launch the selected file with its registered application. I've included the `app.cpp` file in the package and it should compile with most compilers. I originally tested it with the 3.2.2 MinGW g++ compiler and I believe the executable I've included in the package was compiled with the 3.4.5 MinGW compiler, which is what's current on my system (I rarely change compilers). I compressed it with the UPX executable 3.02w compressor.

Appendix 2: launching project files

I use cygwin on my Windows XP system to give me a bash console window where I spend most of my time. If you have the `goto.py` script from http://code.google.com/p/hobbyutil/, you can use it to launch project files whose names are kept in a file. For example, I type the command `pr` in at the command line and I'm presented with a list of current project files to open. Here's my current list:

```
--> pr
 1  Lawn mower notes
 2  Continuity tester
 3  Color-coded tester
 4  Other continuity tester stuff
```

```
 5  Other electronic equipment ideas
 6  Ammeter
 7  Voltmeter
 8  Current source
 9  Logarithmic amplifier
10  RMS

$         Product ideas
ac        Electrical Safety
an        Analytic geometry
comp      Components
compc     Component characterization
etips     Electronic tips
id        Ideas
ide       Electronic ideas
ids       Shop ideas
l         lapp CSV
need      Needed stuff
not       Notable
proj      Electronic projects
sp        Sprinklers
t         To do list
tips      General tips
tips1     Tips, vol. 1
unc       Uncertainty
wart      Wall warts
yard      Yard tasks
Selection?
```

I can choose the top part of the listing by typing in an integer. The bottom part of the listing contains those items that I've defined aliases for, as I open them a lot. Here's a sample of the data file defining this list:

```
# Magazine articles ! mag | d:/p/shop/MagazineArticles.ods
Continuity tester | D:/p/electronics/projects/ContinuityTester.odt
Ideas ! id | D:/p/doc/ideas/ideas.odt
```

The continuity tester listing is numbered because it doesn't have an alias. The Ideas listing is given an alias, as shown by the red text.

One of the most important features is shown by the first line: this is an entry that is commented out because of the leading # symbol. But the "link" to the project file is still in that file -- thus, I can search that file for a particular project I was working on. **I never can remember where the file is or what it was named, but I can quickly find it in the project's text file**. This feature has saved me a lot of time.

Here are the two shell functions I use to launch this stuff:

```
# Function to launch applications.
#   applaunch application     Launches application
#   applaunch a appl          Adds application appl to storage
#   applaunch e               Edits the storage file
#   applaunch -t appl         Check the paths in the file
#   applaunch v appl          Use vi to edit the indicated appl
#   applaunch                 Requests an integer to launch a listed appl
#
# Parameters:
#   1   name (used to make tmp file and rc file)

applaunch()
{
    typeset tmp=$TMP/${1?Need name of rc file}.$$
    typeset dir
    typeset appfile="$CHOME/.${1}rc"
    typeset logfile="$CHOME/.${1}rc.log"
    typeset APP="$bb/app.exe"
    typeset edit=""
    typeset GOTO=_goto.py
```

```
    shift

    if [ $# -eq 0 ] ; then
        # If no arguments were given, use the rc file's entries &
        # prompt the user.
        appl="$($PYTHON $PYTHONPGM/$GOTO -c $appfile 0)"
        [ "$appl" = "" ] && return
    else
        if [ "$1" = "-t" ] ; then
            # Check the paths in the file
            $PYTHON $PYTHONPGM/$GOTO -t $appfile
            return
        fi
        case $1 in
            a|add) # Add application to beginning of file
                echo $($PYTHON $PYTHONPGM/abspath.py ${2?Need a file name})
>$tmp
                cat $appfile >>$tmp
                cp $appfile $HOME/.bup/${1}.$$  # Make a backup copy
                mv $tmp $appfile
                return;;
            e|ed|edit) # Edit the storage file
                $EDITOR $appfile
                return;;
            S) # Search the config file for a string
                grep -i $2 $appfile
                return;;
            v) # Edit the application with vi
                edit="yes"
                appl="$($PYTHON $PYTHONPGM/$GOTO -c $appfile 0)";;
            *)  typeset path="$($PYTHON $PYTHONPGM/abspath.py $1)"
                if [ -r "$path" ] ; then
                    echo "$(dt) $path" >>$logfile
                    $APP -a open "$path"
                else
                    appl="$($PYTHON $PYTHONPGM/$GOTO -c $appfile $1)"
                    echo "$appl $path" >>$logfile
                    [ "$appl" = "" ] && return
                    $APP -a open "$appl"
                fi
                return;;
        esac
    fi
    if [ "$edit" = "yes" ] ; then
        $EDITOR $appl
        return
    fi
    [ "$appl" = "" ] && return
    $APP -a open "$appl"
}

pr()
{
    applaunch project "$@"
}
```

The `pr` command has some options. One of the more useful is the `-t` option, which requests the script to check that each of the files is readable and print out any that are not (e.g., because you've moved or renamed the file.

Another command is `pr a filename` which adds a file to the front of the file (the logic is it's something you're currently working on, so you'd want to see it listed first by being at the front of the file). I use `pr e` to call the file up in my editor. The most frequent use is to type `pr` to be prompted for which file I want to open or e.g. `pr t` to open my todo list (an alias or number can be typed on the command line and the relevant file will be opened without prompting). The `pr S` command lets you search the data file for a regular expression and prints out any matches in color; it uses `grep` to

do this, so you'll need to be in a UNIX-like environment (the GNU `grep` command will do color-highlighting when it's configured properly and this is a very useful feature).